Roboskeleton: An architecture for coordinating robot soccer agents

David Camacho[a,*], Fernando Fernández[b], Miguel A. Rodelgo[b]

[a] *Ctra. de Colmenar Viejo, Universidad Autónoma de Madrid, 28049 Cantoblanco, Madrid, Spain*
[b] *Avda. de la Universidad 30, Universidad Carlos III de Madrid, 28911 Leganés, Madrid, Spain*

Abstract

SkeletonAgent is an agent framework whose main feature is to integrate different artificial intelligent skills, like planning or learning, to obtain new behaviours in a multi agent environment. This framework has been previously instantiated in a deliberative domain (electronic tourism), where planning was used to integrate Web information in a tourist plan. RoboSkeleton results from the instantiation of the same framework, SkeletonAgent, in a very different domain, the robot soccer. This paper shows how this architecture is used to obtain collaborative behaviours in a reactive domain. The paper describes how the different modules of the architecture for the robot soccer agents are designed, directly showing the flexibility of our framework.

Keywords: Multi agent architectures; Robot soccer; Distributed AI systems

1. Introduction

The intelligent agents and multi-agent systems (MASs) technologies (Brenner et al., 1998; Wooldridge and Jennings, 1994) have provided new solutions to deal with complex problems in several domains such as data mining, e-commerce, medicine, stock market, intelligent manufacturing control, simulation of complex societies, etc. The MAS techniques provide a natural way to design and implement decentralized and distributed systems where each element, or agent, can take its own control decisions and modify its behaviour using the environment information or its previous experience.

Two main technologies have been applied in the development of intelligent distributed systems, MASs and holonic systems[1] (Shen and Norrie, 1999; Leeuwen and Norrie, 1997). Both approaches have similar goals: to develop systems based on software modules (usually named software agents) that implement different control strategies (e.g. routing selection, scheduling, planning) to take a decision, and that can use other techniques like cooperation or negotiation to achieve a particular goal. The main goal of both approaches is to increase the adaptability and robustness of the systems. Both approaches can be characterized by concepts like autonomy, pro-activeness, coordination, and language communication to obtain intelligent systems which are able to adapt to the environment. Other characteristics such as social organization, cooperation, knowledge representation, coordination, or negotiation, are necessary to build complex societies of agents. The differences among both approaches are related on how the information is managed. In the holonic systems there is an explicit division between the physical and information management, whereas in the MAS field this distinction does not exist. Other characteristics, like recursivity, is very typical of holonic systems, just as mobility is usual in MAS technologies. Holons always cooperate to solve a problem, however the agents in a MAS can cooperate, or compete to achieve their particular goals.

Due to the complexity to build these distributed systems, it is very common to use some kind of framework that aids engineers to implement the agents and their desired behaviours. Those frameworks provide some kind of basic agent with a pre-defined internal architecture, that can be modified by the programmer. However, most of the architectures are not very flexible and it is complicated to use this agent architecture in a wide range of domains (i.e. deliberative and reactive). This paper describes a generic multi-agent framework, called SkeletonAgent, capable of integrating artificial intelligence (AI) classical techniques to build intelligent systems. This framework has characteristics learned from other agent-based models like CooperA (Sommaruga et al., 1996), and from multi-agent models like ABC^2, or Retsina (Paolucci et al., 1999; Decker and Sycara, 1997). Our approach has extended previous models so that AI techniques (like planning and learning) can be applied in different domains like Web information gathering (Knoblock and Ambite, 1997) or Genetic Programming. The paper briefly describes the SkeletonAgent architecture (Camacho et al., 2005). The main goal of the paper is to deal with its instantiation into the robot soccer domain, i.e. how to build a team of robot soccer agents that can be used in the RoboCup simulation league (Kitano et al., 1997). This domain uses reactive agents where decisions of the players are based mainly on immediate sensorial information. The control scheme of the agents follows a REactive Controllers based Cooperative Architecture (RECCA) (Fernández et al., 2000), which is a distributed scheme to achieve the collaboration of soccer agents.

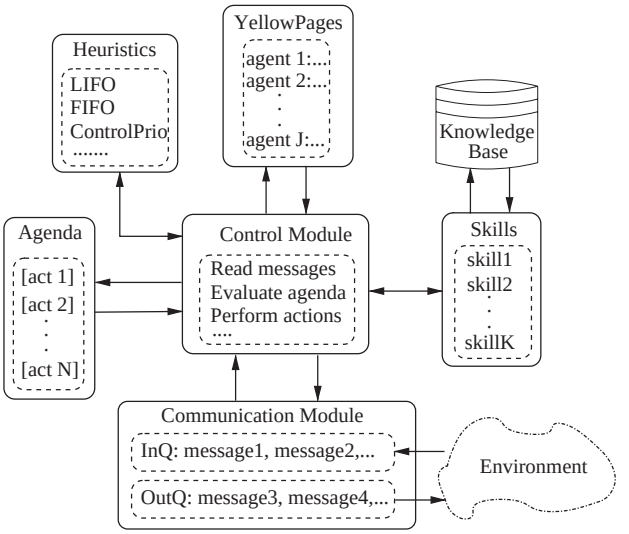
The paper is structured as follows: Section 2 describes the architecture of a generic software agent, and the related multi-agent framework (SkeletonAgent). Section 3 describes the robot soccer domain, reviewing some agent architectures implemented in that domain, and describing RECCA. Section 4 provides the instantiation of SkeletonAgent in the RoboCup Simulation League. Finally, Section 5 summarizes the conclusions of the research.

2. SkeletonAgent

In this section, we briefly describe our software agent framework, where we address the main characteristics of the agent and multi-agent architecture defined by this framework.

2.1. Agent architecture

Agents in SkeletonAgent are composed of several software modules (Camacho et al., 2005). Fig. 1 shows those modules and their interconnections.



- *Agenda/skills*. The agenda is a dynamic structure that stores items called *acts* which are directly related to a skill of the agent. These acts represent the actions that the agent is considering to select and carry out at a given moment. The content of these messages depends on the current implementation domain. For instance, in the RoboSoccer domain, selecting the act "kick the ball" from the agenda will perform the agent skill (a routine) that carries out this act. As explained before, an agenda contains acts. Some of them can be decomposed into lower-level acts, which are subsequently introduced into the agenda. When a particular act (atomic, or low level) cannot be decomposed further, it will be executed by the agent. Those executable acts are actually the skills of the agent. Automatic access to the Web, or executing a planner are examples of skills. The SkeletonAgent agent architecture provides a simple mechanism to use any AI module as a new skill. If this new skill cannot be decomposed it is simply launched as an atomic act. Moreover, if the skill can be decomposed in several sub-acts, these can be independently managed in the agenda (Camacho et al., 2005).
- *Heuristics/control module*. Agents have a set of heuristics that are used to decide at any time what act to select from the agenda. SkeletonAgent provides some standard behaviours like: last in/first out (LIFO), first in/first out (FIFO), or a *ControlPrio* policy which allows the selection of the oldest acts, so that they can eventually be executed. These policies have to be selected before running the agent and remain fixed during execution time.
- *Knowledge base*. This module stores the knowledge that can be used by the agent skills. The architecture

does not provide a language for representing generic knowledge. The programmer can use any structure that is required (files, databases, variables, etc). The actual knowledge stored depends on the domain. Examples for the RoboSoccer domain will be given in Section 4.2.

- *Yellow pages*. This module stores the knowledge that an agent has about all the agents belonging to its team. This information consists of a list made up of the name of its partners, and the name of the skills they can accomplish. Any skill can be considered as an abstraction of an action that will be accessible to other agents in the team. In fact, it means that the agent has meta-knowledge about itself (through its skills definition) and its partners.
- *Communication module*. Agents in SkeletonAgent use a communication language to share information. The control module decides when to send messages, and it is also the module which receives messages from other agents. Once the control module has received a message, it can be distributed to any other module of the agent. The communication process is implemented by three interconnected submodules: the *communication manager module*, that manages the messages received from the language module in the input/output queue; the *language module*, that translates the internal data (used by the agent) into the communication language used by the agent (i.e. an standard KQML message) or vice versa; and finally, the *communication module* that is responsible for serializing and deserializing the information, so the message can be sent (received) through (from) the physical network.

There are a variety of multi-agent frameworks for designing intelligent agents and behaviour models. Although they have different focus of interest, all these frameworks provide coherent, high-level views of intelligent agency. However, much of the complexity of building intelligent agents is in the low-level details, especially when building agents that exhibit high degrees of competence while interacting in complex environments. The flexibility of SkeletonAgent arises from its modular architecture conception and implementation of both the agents and the relationship among them. Any agent is designed using a set of software components, that are specified in the design phase. Although it is possible to use the pre-defined modules provided by the framework, any of them can be modified or replaced. This allows to build MAS for any kind of domain (deliberative or reactive). Other architectures, and frameworks, like Jade, JATLite, ZEUS, JAFMAS, etc. provide some kind of *agent template* (classes, libraries, etc.) that can be used to implement the agent. However, the modification of its internal behaviour (and components) is usually a hard task.

2.2. Multi-agent architecture

In this section, how to implement societies of agents by means of the model outlined in the previous section is described. Our architecture, SkeletonAgent, provides a reusable code to help with the development of different kind of MASs. The main goal of this framework is to easily allow the integration of AI solving problem techniques (like planning or machine learning) (Camacho et al., 2005; Aler et al., 2003) in both deliberative and reactive domains. There are three main roles in the system: users, solvers, and information agents. Therefore, our system follows a three-layer architecture, as other approaches like Retsina (Paolucci et al., 1999; Decker and Sycara, 1997). As we also want to implement teams of agents, a new kind of agents has been included (control agents). Every team is managed by a specialized agent named CoachAgent (*CCH*). It is necessary to use a single ManagerAgent (*MNG*) to manage the different teams. The flexibility of the coordination and cooperation approach will allow to our agent-based systems to modify their behaviours and adapt their capabilities to the main characteristics of the domain. Therefore, in SkeletonAgent (like other MAS frameworks), any system implemented needs at least the following agents to work properly:

- *Control agents*. They manage the different agents in the system. There are two types of them:
  - ManagerAgent (*MNG*). This agent is similar to *AMS* agents (as in FIPA). Its main roles are to add and remove other agents from the system, and to control which agents are active in the agent society. This agent is responsible for building a team of agents. To do this, when any agent requests to be inserted in the society, the *MNG* determines which teams require this agent.
  - CoachAgent (*CCH*). They control a team of agents, guaranteeing stability and smooth cooperation of the active agents. The CoachAgents report problems directly to the *MNG* (for instance, when a new agent is required for the team). These agents are also used to guarantee that the yellow pages of the team members are coherent.
- *Execution agents*. These agents are responsible for achieving the different goals of the system. To coordinate different teams of agents it is possible to include a new skill in the control module of the agents. Currently, there exist different kind of execution agents; we have implemented agents which are able to use a planner to solve problems (PlannerAgents), information agents that can retrieve Web data (WebAgents), or agents that can interact with the RoboSoccer simulator.

In summary, the main characteristics of a MAS that can be implemented within this framework are:

- Agents in the system use message-passing to communicate with other agents.
- All the agents have the same architecture and they are specialized in different tasks through the implementation of different skills.
- Although the communication language is the same for all the agents in SkeletonAgent, it is possible t o distinguish two different types of communication messages. On the one hand, there are control messages whose main goal is to manage the behaviour of the system (control communication module). On the other hand, execution messages are used to share knowledge and tasks among the agents, to achieve desired goals (execution communication).

To start the MAS correctly, it is necessary to perform the following steps:

(1) First, the *MNG* is executed.
(2) Agents in the system need to register themselves with the *MNG*. Once a *CCH* has registered, the *MNG* will select the necessary execution agents from its white pages and will build an operative team. If there are not enough agents, the *CCH* will wait for them. To build a team the *MNG* selects the execution agents and provides the necessary information to the *CCH*. Once the information of the agents has been stored in *CCH*'s yellow pages, it updates the yellow pages of its execution agents. To select the necessary agents to build a group, the *MNG* uses the Ontology of the *CCH* agent.
(3) Once a team is built, the execution agents can only communicate with the agents belonging to its team or with its *CCH*.

3. RoboSoccer domain

The next subsections introduces the RoboCup domain, as well as the control algorithm used for the RoboSoccer agents. Then, the design of this MAS, as well as the control algorithm will be provided in the next section.

3.1. RoboCup and MAS

The Robot World Cup Initiative (RoboCup) (Kitano et al., 1997) provides a standard problem, soccer, for the research of AI and intelligent robotics. Added to the real robot leagues, RoboCup provides a software platform, called Soccer Server simulator, for research and simulate several problems which appear with real robots. The Soccer Server is an environment for confronting two teams of players (software agents) (Noda, 1999) that are controlled by several types of systems. A match is carried out in a server-client style: a server, Soccer Server, provides a virtual field and simulates all movements of a ball and players. The clients become the players' brains and control their movements. Communication between the server and each client is done via UDP/IP sockets. Therefore, users can use any kind of programming system which has UDP/IP facilities. In another level of the communications, different protocols are developed between the clients and the server to transmit sensor information and agent control commands.

Given this platform, several architectures have been applied to control a RoboSoccer team. In Matellán and Borrajo (2001), an agenda-based architecture is used to integrate the different skills of the players, each of them composed of different basic actions such as kick the ball, or send a message to another player. However, when integrating these actions in order to obtain complex behaviours, the designer must decide on a number of hierarchical levels in the control architecture, as defined in Stone (2000). This decision depends on the knowledge about the model of the domain. If the model is very well known, it is possible to obtain a very detailed description of each task, so the domain, and how to solve the tasks in it, can be deeply described and hierarchized. When the model is very dynamic, stochastic, and unknown, as in the case of RoboSoccer, the development of this hierarchy of tasks is not easy to create, given that the model of the domain must be created dynamically while interacting with it, and how to integrate pre-defined knowledge with the models being learned is not an easy task. Thus, reactive architectures are very robust in domains where a description of the model is not known, and are hard to build. However, to coordinate reactive systems or reactive agents, a centralized reasoning is typically used, which indicates to each agent the task that it has to solve. However, in domains like the RoboSoccer, a centralized reasoning cannot be implemented, because it requires a perfect knowledge of the domain, and a high communication capability that is not provided by the soccer server simulator. A simple way to control the team members and to obtain a collaborative behaviour among the agents can be achieved through the implementation of the RECCA which is described next.

3.2. RECCA

The goal of this control architecture, designed for RoboSoccer agents, is that the different players of the team are able to keep a team formation (Fernández et al., 2000). This formation defines the number and location of the players in the field, for instance, a 4-4-2 formation (four defenders, four midfield players, and

two forwards) or any other formation (4-3-3, 5-3-2, etc.). The main characteristic of this approach is that the formation is kept only by using local information. This means that the players will play without knowing the absolute locations of other players, so they will only use the visual information that they receive in each moment, minimizing the communication among players. So it is desirable that the players maintain the formation whatever the situation of the game is, i.e. if they own the ball, if they are defending, etc. The architecture is based on three elements. Firstly, a formation type (for instance, 4-4-2, 5-2-2, etc.) is defined. The team must follow that formation, which can be modified by the coach of the team, but it is supposed that it will not change many times in a match.

The second component to maintain the formation of the team is the global leader. The global leader is the player that owns the ball, or that is supposed to be able to obtain it, and it is the only global information that all the agents must share. The global leader of the team changes in time, and is the centre of the play for its team. So, if one player is the global leader, all the other players will play using it as the central point of the team. The decision about who the global leader is, is defined by a reduced communication protocol. If a player is very close to the ball but it is not the global leader, it can send a message to the rest of the team members informing that he is the new leader, obtaining the ball. Furthermore, a global leader may give the leadership to another team member by, for instance, passing him the ball.

The last element of the architecture is the player role. Each player plays a role in the team. This role is defined by three elements: (i) the play area or absolute positions where the player can be located; (ii) the local leader of the role; and (iii) the soccer role, i.e. whether the player is a defender, midfield player, or an attacker. Fig. 2 shows the play areas and local leaders for a team with a 4-3-3 formation. The play area of each player is defined by the rectangles, and the local leaders are defined by arrows. The global leader (which owns the ball) is the only player who does not have a local leader.

The play area associated to each player is defined "a priori", and depends on the formation. These areas must be big enough to allow the players to cover the field and to follow their local leader, but small enough to maintain the coherence of the team. The local goal of each player is to follow his local leader. The idea is that if the players are able to follow their local leader, they will keep the formation of the team. Depending on the definition of the local leaders, the location of each player in the field may change. The definition of the local leader of each player depends on the global leader. This means that depending on who the global leader is, the local leader of each player may be different. This relationship among global and local leader is pre-defined with the play strategy and the formation of the team. For

Fig. 2. Local areas and leaders.

instance, given the situation of Fig. 2, let us assume that the player that owns the ball decides to pass to the player in its left. Then, the receiving player becomes the global leader. Based on the new global leader, all the players updates their local leaders to the situation defined in Fig. 3. In the figure, the leadership relationships that have changed are bolded. For instance, the new global leader becomes also the local leader of the past global leader. In the same way, it becomes the local leader of all the midfield players.

The relationship between the local leader and the global leader has been defined by hand, but optimal configurations for different formations could be learned. In Reynolds (1999), the concept of leader is also used when implementing following leader behaviours is a game. In that case, the goal is to provide some characters with the capability of following a leader without crowding and staying out of the leader's way. In our case, leader following requires the same capabilities, but it is also constrained by vision limitations.

The control algorithm of all the players (except the goal keeper, which is implemented differently) is defined in Fig. 4. It shows that all the information used to take the decisions is local to each player. The only global information is which player is the global leader, so it is shared by all the team. Each of the actions shown in the flow represents high-level acts that can be executed by the agents. For instance, the first action that all the players do is to verify if they are blocked by another player, i.e. if another player is in its way, and to unblock the situation. There are several different acts defined. For instance, "look for the ball" is an act where the player rotates to find the ball. Acts like "go to the ball", "get closer to my local leader", or "go to the centre of my play area" makes the agent move to different

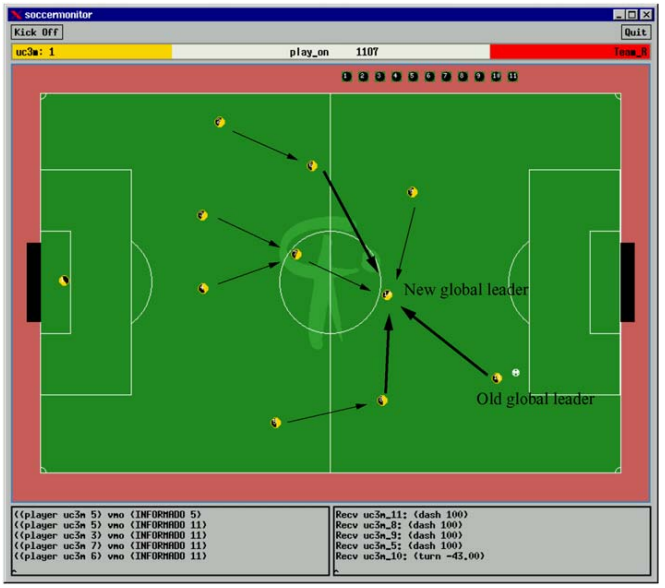


Fig. 3. Local areas and leaders.

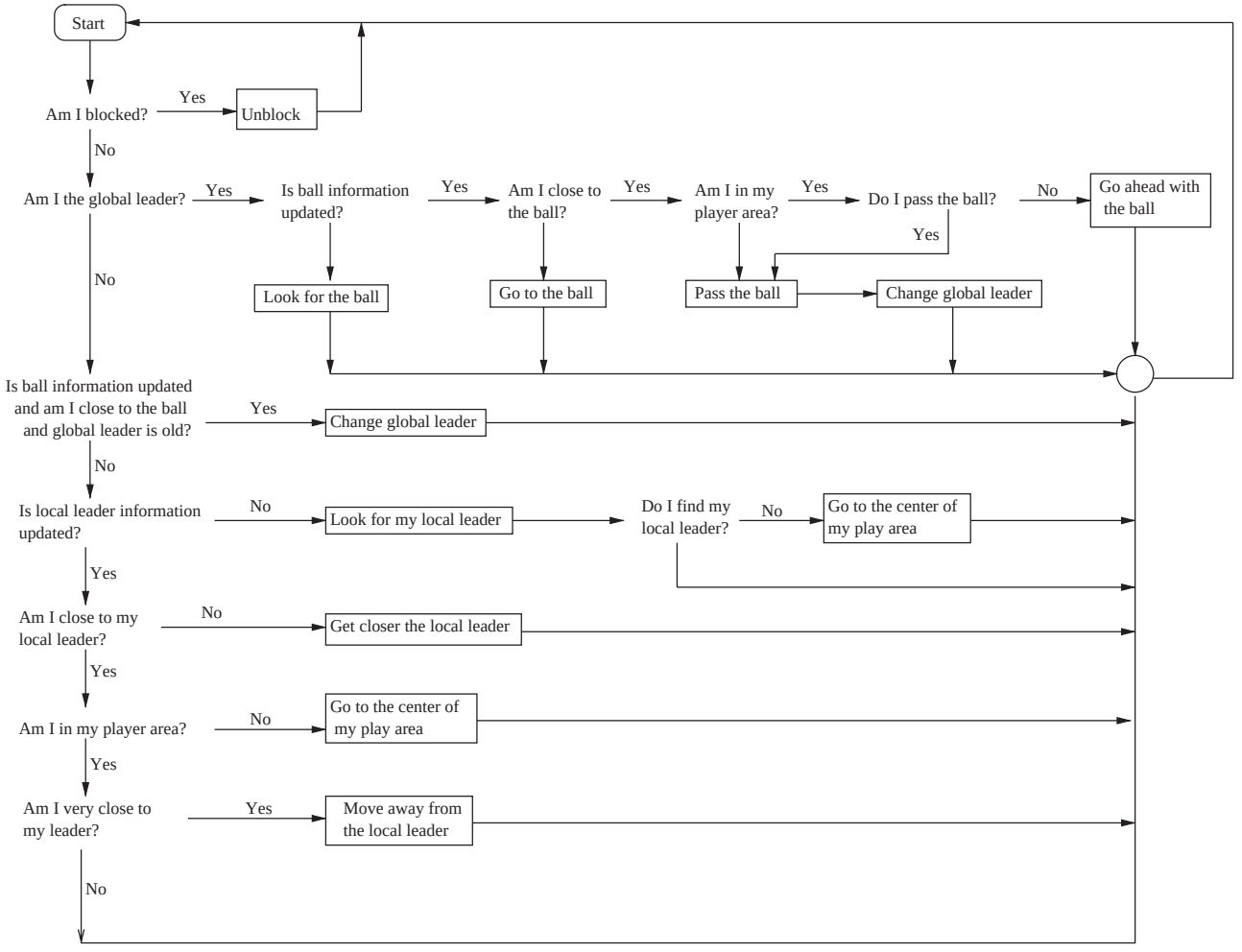


Fig. 4. RECCA control algorithm for players.

locations. "Change global leader" and "look for the local leader" are the only two communication acts. In the first one, the new global leader is communicated to the rest of the players. In the second one, a player ask for the position of its local leader, so its local leader, or any other player who knows his position, may answer this question.

The control scheme has two main branches. The first one, if the player is the global leader, where it tries to go forward or pass the ball to a partner. The second one, if the player is not the local leader, where it tries to stay in its play area, following to its local leader. Each of the actions shown in the flow could be studied to improve the behaviour of the team so, for instance, if a player is not the global leader, instead of keeping in the centre of its play area, he could search for a good position to receive the ball, etc.

4. RoboSkeleton: design of a team of RoboSoccer agents

This section describes a specific instantiation of SkeletonAgent in the RoboCup domain, which we have called RoboSkeleton. It is a multi-agent team of software robots (reactive agents) that allows us to implement different control strategies for playing soccer.

4.1. Multi-agent characteristics of RoboSkeleton

The general multi-agent architecture that supports SkeletonAgent was introduced in Section 2.2, showing three main kinds of agents: manager agents, coach agents and execution agents. Fig. 5 shows how this architecture has been adapted to the RoboSoccer domain.

Manager agent appears to control the number of agents in the system, taking into account the restrictions

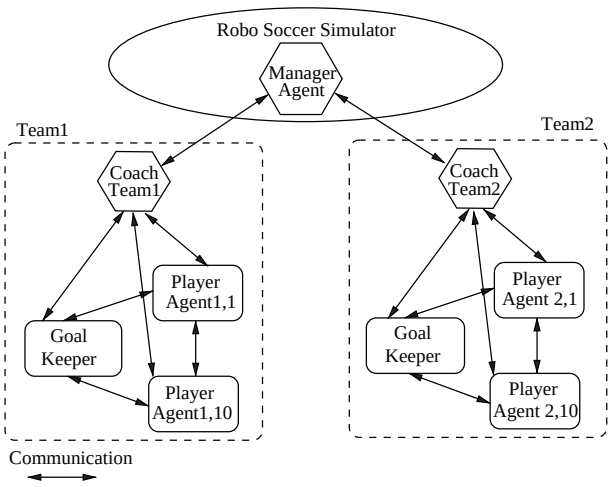


Fig. 5. RoboSkeleton multi agent architecture.

of this domain: only two teams, each of them composed of 11 players. In the case of implementing robot soccer teams in the Soccer Server simulator, this manager agent is implemented by the simulator itself, which is the element to which the different agents of the different teams must connect. Nevertheless, Soccer Server simulator implements all the roles defined in Section 2.2 for this kind of agent, i.e. to add and remove other agents from the system (one coach and 11 players per team), control which agents are active in the agent society and group agents in teams.

A coach agent manages the agents of a team, guaranteeing stability and smooth operations of the active agents, as defined in Section 2.2. Therefore, in the RoboSoccer domain, this agent must define the play strategy of the team, the role that each player must play, etc. In the robot soccer domain, coaching is demonstrating a very important impact in the results obtained by the teams (Riley et al., 2002). We suppose that our coach has three main capabilities, following the ideas introduced in Riley and Veloso (2001). The first one is modelling opponent behaviour in order to characterize it from a pre-defined set of different possible opponent models. The second one, and once opponent behaviour has been characterized, is deciding the right play strategy. And lastly, communicating the new strategy to the players. This last capability is the only one implemented up to now in our software, but we will show how the architecture should work with the other ones.

Execution agents are responsible for achieving the different goals of the system, that in the RoboSoccer domain is defined as winning the match. In this case, RECCA defines two different kind of agents, the goal keeper and the rest of the players, i.e. defender, midfield, and forward players. The role of each player depends on the coach, so it does not need to be defined at this level.

4.2. Characteristics of RoboSkeleton agents

This section provides a detailed description about the specific characteristics of each agent in the RoboSkeleton system. Table 1 describes the specific agent characteristics.

- *Control agent: ManagerAgent* (*MNG*). As we have stated before, in the RoboCup application of the multi-agent SkeletonAgent architecture, this agent is implemented by the Soccer Server simulator, so it has not been implemented following the standard agent architecture.
- *Control agent: CoachAgent* (*CCH*). The acts that this agent manages are related to defining and communicating the strategy to the players. This communication can be defined at the beginning of a match, but acts for identifying the strategy of the opponent, and

Table 1
Specific attributes in the RoboSkeleton agents

	Manager agent	Coach agent	Goal keeper	Player
Agenda		The following acts are defined: • Communicate the strategy to the players • Extract opponent model • Define new play strategies from opponent models	The following acts are defined: • Look for the ball • Locate in goal • Catch the ball	The following acts are defined: • Look for the ball • Look for the leader • Change leader • Others
Heuristics/ control module		FIFO/sequential	FIFO	RECCA
Skills		It can communicate with the player agents to inform about changes in play parameters, like play strategy, size of the play areas, etc.	Low level skills to catch the ball and to kick it to another team member	All the skills related with playing soccer, as well as capabilities to connect to the other agents
Knowledge base		Opponent models, play strategies, information relating opponent models and successful play strategies		Play strategies, global and local leader information
Yellow page				
Communication module		Soccer Simulator Language and UDP/IP protocol	Soccer Simulator Language and UDP/IP protocol	Soccer Simulator Language and UDP/IP protocol

from that model, defining a new strategy, are included. The knowledge base introduces two kind of information. On the one hand, the information required to model the behaviour of the opponent. On the other hand, the information required to define the play strategy from the opponent model. Players are very homogeneous, so no yellow pages are required, given that the rest of agents will be treated as equal. Communication module follows the ideas of SkeletonAgent, establishing the three communication levels.

- *Execution agent: Goal keeper*. The goal keeper has an independent implementation from the rest of the players. His only behaviour is to keep in his goal, and to move only when a forward player kicks the ball to the goal. This last action can be hand-made programmed or learned with machine learning methods, as defined in Fernández and Borrajo (2000). Knowledge managed is limited to sensorial information.
- *Execution agent: Player*. There are three kinds of skills, that compose a hierarchy of skills. The first one is a basic skill that corresponds with an action executable in the simulator, like the *turn* and *move* soccer server commands. The second one is a single skill, that is composed of a sequence of basic skills, but that does not require external information to be executed, for instance, a sequence of *dash* commands. The third level is composed of skills that can involve complex behaviours, and can be composed of basic and single skills, and decisions can be taken depending on additional information of the agent, the server, etc., that can be received while the skill is being executed. The last two kinds of skills are the ones defined in the RECCA control architecture shown in Fig. 4.

  Even though the players follow a reactive behaviour, some information is managed: on the one hand, information computed from the information received through the sensors; on the other hand, information required to execute the RECCA control scheme, such as the size of the play area and the relationship among global and local leaders, that are given by the play strategy. Furthermore, global leader is the only knowledge shared by the team, and it requires a communication process among the agents. Players are homogeneous, so no yellow pages are required.

Communication among the agents can follow the three layers architecture defined for SkeletonAgent in Section 2.1. Thus, the UDP/IP protocol is used in the communication layer; the Soccer Server communication language is used in the language layer; and each agent has its own communication manager Module, defined to process all the messages received and that it needs to send to other players. Furthermore, with the Soccer Server simulator, all communications are centralized by the server because any other kind of communication is forbidden by RoboCup simulation league rules.

4.3. RoboSkeleton validation

To evaluate the RoboCup team implemented, several factors can be taken into account, both from the quantitative (number of goals scored, lost balls,

successful passes, etc.) and the qualitative point of view (individual and team coordination, play strategy execution, etc.). The goal of this work is to show that SkeletonAgent can be successfully applied on the robot soccer domain so, in this case, we have considered evaluating the qualitative factors that define the control module of the RoboSkeleton agents, given that our main interest is, in this case, to verify the capability of the architecture to execute coordinate global behaviours, and not to develop high quality low-level ones.

Section 3.2 introduced the fact that the main goal of RECCA is that the soccer players are able to keep a play strategy, for instance, 4-4-2, or any other one. This is a global parameter that must be defined. Other three local elements were defined for each player: the role, the local leader, and the size of the play area associated to the player. From these four elements, we have performed several experiments to define the global one, i.e. the play strategy, and one of the local ones, the play area of each player. The goal of this experimentation is to define which is the more accurate one. This accuracy is obtained visually by the designer, by observing if the formation of the team is kept, and if each player is able to follow his local leader, or if the local leader is lost, and hence, formation is also lost.

4.3.1. Defining the play strategy

For defining the play strategy, the structure 4-4-2 (four defenders, four midfield players, and two attackers) is used as a basic play strategy. This basic team is faced against other teams with different play strategy. In the first evaluation, the basic team plays against another with a 5-5 strategy, i.e. only two lines of players. When these two teams are playing, it was observed that while the basic team was able to keep the play strategy, the other team was not. The main problem appears when the players try to follow their local leader when both are located on the same play line (given that there are only two lines). So, the player places himself in his line horizontally with respect to his leader, while attacking or defending are actions that typically must be executed vertically in the field. Then, the opposite case was tested, executing several matches of the basic team against another with a 4-3-2-1 play strategy. Even though this team maintains the team structure better than the 5-5 one, the increment in the number of lines increases the effort to maintain the team strategy. The 4-4-2, as in the previous case, is able to keep the formation of the team for a longer time. Therefore, we can conclude that the RECCA control architecture does it better when a 4-4-2 play strategy is followed, even though it can be instantiated with different ones.

4.3.2. Defining the play area size of the players

For tuning the play area, the basic team tested in the previous test, with the 4-4-2 play strategy, is used. It is tested playing with different teams with the same play strategy but with different sizes of the play area of the different players:

- In the first experiment, the original team plays against another, where the play area of each player has been homogeneously increased. This new team seems to maintaining the cohesion of the team better, given that the player can stay closer to each local leader, independently of the position of the global leader.
- In the second experiment, instead of increasing the play area of all the players homogeneously, the ones of the attackers and the defenders have been increased in a higher proportion than the midfield player ones. At the same time, the midfield areas have been modified to more accurate areas that try to minimize the effort of the team to keep the play strategy.

We can conclude that increasing the team area improves the capability of each player to follow its local leader, given that they can move in a higher area of the field. However, it can make the players keep very close among themselves, localizing all of them in a small area, which cannot be good when playing real matches. This element shows the difficulty of defining a metric that allows us to empirically measure the goodness of a strategy, or certain parameters such as the area size of the players, and this is the reason we have not used it in this work. Testing them with real teams would require a finer tuning of low-level skills that have not been carried out up to now.

5. Conclusions

This paper has presented the instantiation of SkeletonAgent, a flexible architecture for building MAS, in the robot soccer domain. This instantiation, named RoboSkeleton, implements a team of agents that can be executed in the RoboSoccer domain, following a reactive behaviour. With the adaptation of the architecture to this new domain, we show the flexibility of our approach, and how it can be adapted following the requirements of very different application domains.

The RoboCup domain is an interesting domain where the SkeletonAgent architecture can be successfully used to design the different agents of the MAS. For instance, in relation to the multi-agent architecture, the RoboSoccer domain has shown it requires a manager-coach-execution agent scheme, related to the RoboSoccer coach and players, respectively. Furthermore, the agent architecture is useful too, even though sometimes not all the modules are required.

In the future, several topics will be addressed with RoboSkeleton: the tuning of the skills and integrating

new abilities in the players will result in more sophisticated behaviours, and hence, will allow stronger comparisons with other teams. Our modular conception of SkeletonAgent allows to apply it into very heterogeneous domains: currently we are using this architecture in two, a genetic programming (Aler et al., 2003) domain and in a distributed searching of electronic news (Camacho and Aler, 2005) domain. In the future, we would like to apply our architecture in new domains such as the intelligent manufacturing (Shen and Norrie, 1999).

References

Aler, R., Camacho, D., Moscardini, A., 2003. In intelligent agent software engineering. Cooperation Between Agents to Evolve Complete Programs, Valentina Plekhanova. University of Sunder land, United Kingdom. Idea Group Publishing, pp. 213 228.

Brenner, W., Zarnekow, R., Wittig, H., 1998. Intelligent Software Agents. Foundations and Applications. Springer, New York ISBN: 3 540 63411 8.

Camacho, D., Aler, R., 2005. Software and performance measures for evaluating multi agent frameworks. Applied Artificial Intelligence: An International Journal 19(6), 645 657.

Camacho, D., Aler, R., Borrajo, D., Molina, J.M., 2005. A multi agent architecture for intelligent gathering systems, AI Communications. The European Journal on Artificial Intelligence 18 (1), 1 19.

Decker, K., Sycara, K., 1997. Intelligent adaptive information agents. Journal of Intelligent Information Systems 9, 230 260.

Fernández, F., Borrajo, D., 2000. VQQL. Applying vector quantiza tion to reinforcement learning. In: RoboCup 99: Robot Soccer World Cup III, no. 1856 in Lecture Notes in Artificial Intelligence. Springer, Berlin, pp. 292 303.

Fernández, F., Gutiérrez, G., Molina, J.M., 2000. Coordinación global basada en controladores reactivos en la RoboCup. In: Workshop Hispano Luso en Agentes Fisicos, pp. 73 86.

Kitano, H., Tambe, M., Stone, P., Veloso, M., Coradeschi, S., Osawa, E., Matsubara, H., Noda, I., Asada, M., 1997. The Robocup synthetic agent challenge. In: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI97). San Francisco, CA, pp. 24 49.

Knoblock, C.A., Ambite, J.L., 1997. In software agents. In: Bradshaw, J. (Ed.), Agents for Information Gathering. AAAI/MIT Press, Menlo Park, CA.

Koestler, A., 1967. The Ghost in the Machine. Arkana Books.

Leeuwen, E.V., Norrie, D., 1997. Intelligent manufacturing: holons and holarchies. Manufacturing Engineer 75 (2), 86 88.

Matellán, V., Borrajo, D., 2001. ABC^2 an agenda based multi agent model for robots control and cooperation. Journal of Intelligent and Robotic Systems 32 (1), 93 114.

Noda, I., January 1999. Soccer Server Manual, version 4.02 ed.

Paolucci, M., Kalp, D., Pannu, A.S., Shehory, O., Sycara, K., 1999. A planning component for retsina agents. In: Lecture Notes in Artificial Intelligence, Intelligent Agents VI.

Reynolds, C.W., 1999. Steering behaviors for autonomous characters. In: Proceedings of Game Developers Conference, pp. 763 782.

Riley, P., Veloso, M., 2001. Coaching a simulated soccer team by opponent model recognition. In: agents2001, pp. 155 156.

Riley, P., Veloso, M., Kaminka, G., 2002. An empirical study of coaching. In: Asama, H., Arai, T., Fukuda, T., Hasegawa, T. (Eds.), Distributed Autonomous Robotic Systems, vol. 5. Springer, Berlin, pp. 215 224.

Shen, W., Norrie, D., 1999. Agent based systems for intelligent manufacturing: a state of the art survey. Knowledge and Informa tion Systems, an International Journal 1 (2), 129 156.

Sommaruga, L., Avouris, N.M., Liedekerke, M.V., 1996. The evolution of the CooperA platform. Foundations of Distributed Artificial Intelligence. Wiley, London, pp. 365 400.

Stone, P., 2000. Layered Learning in Multiagent Systems. MIT Press, Cambridge, MA.

Wooldridge, M.J., Jennings, N.R., 1994. Agents theories, architec tures, and languages: a survey. In: Proceedings of the 1994 Workshop on Agent Theories, Architectures, and Languages. ECAI 94.